Compositional, Expressive, Performant: Choose 3

Improving DB interaction reusability with Sloppy

$ whoami

David Nicponski

SeamlessDocs CTO ‘16-’19

Twitter ‘14-’15

Google ‘06-’14

Limewire ‘05-’06

BearShare ‘01-’04

C++ -> Scala
15 years in OO land
Convert to FP in 2014

Twitter: @**virus_dave**

Presentation Overview

1) The problem
2) Intro to Slick
3) Reusability
4) Examples (Show me the code!)
5) The gotchas
6) Summary
7) Fin

Presentation Overview

1) **The problem**
2) Intro to Slick
3) Reusability
4) Examples (Show me the code!)
5) The gotchas
6) Summary
7) Fin

SQL: This is why we can’t have nice things

SQL has well-known shortcomings, but is the de facto standard for DB interaction

Presentation Overview

1) The problem
2) **Intro to Slick**
3) Reusability
4) Examples (Show me the code!)
5) The gotchas
6) Summary
7) Fin

Slick

Scala DB library

FRM

Scala eDSL for queries and statements

The Slick Idea

An FP-inspired modeling of databases and result sets.

The Slick Idea

Slick provides a **collections-like** DSL to compose operations on conceptual streams of results.

The DSL should be familiar and intuitive to people familiar with Scala.

The Slick Idea

Slick provides a **collections-like** DSL to compose operations on conceptual streams of results.

The DSL should be familiar and intuitive to people familiar with Scala.

The Cast

The Cast

Future[R]

Your favorite lawless ‘monad’ and mine, this eventually (hopefully) contains your materialized results.

There’s also a streaming variant on this.

The Cast

DBIOAction[R, blah, blah] a.k.a **DBIO[R]**

A **Free** monad by another name. Expresses entire DB operations that have not yet been executed.

dbDriver.run(dbioaction) gives us back a **Future** of our result set (or a stream thereof).

The Cast

Query[Rep[R], R]

C'est non une Monad!

The Cast

Query[Rep[R], R]

Operations on the contents of queries work over **lifted types** (**Rep[R]**) which represent DB-side values, not locally materialized values

The operations are reified into an AST representing to computation so far

When eventually compiled and run, the query produces actual values of type **R** for rows in the result set

Slick example

```
val coffees = TableQuery[Coffees]

val q = coffees.filter(_.price > 8.0).map(_.name)
//                          ^       ^          ^
//                Rep[Double]  Rep[Double]  Rep[String]
```

Slick example

```
val q1 = coffees.filter(_.supID === 101)
// compiles to SQL (simplified):
//   select "COF_NAME", "SUP_ID", "PRICE", "SALES", "TOTAL"
//     from "COFFEES"
//     where "SUP_ID" = 101

val q2 = coffees.drop(10).take(5)
// compiles to SQL (simplified):
//   select "COF_NAME", "SUP_ID", "PRICE", "SALES", "TOTAL"
//     from "COFFEES"
//     limit 5 offset 10
```

Presentation Overview

1) The problem
2) Intro to Slick
3) **Reusability**
4) Examples (Show me the code!)
5) The gotchas
6) Summary
7) Fin

Reuse

“Simple” composition still tends to happen at the **DBIOAction** level, since that’s monadic and easy to work with.

But this usually implies sequential DB round trips!

Reuse: Solution

Make it the caller's problem!

```
(Query[A],
 Rep[A] => (my inputs)
) =>
   Query[(A, something)]
```

Reuse: Solution

```
(Query[A],
 Rep[A] => (my inputs)
) =>
   Query[(A, something)]
```

Given an intermediate query of arbitrary type, and a way to extract stuff we care about from that arbitrary type (supplied by the caller), we can produce something useful to add to the query's result type.

Also works for filtering, etc

Reuse: Solution

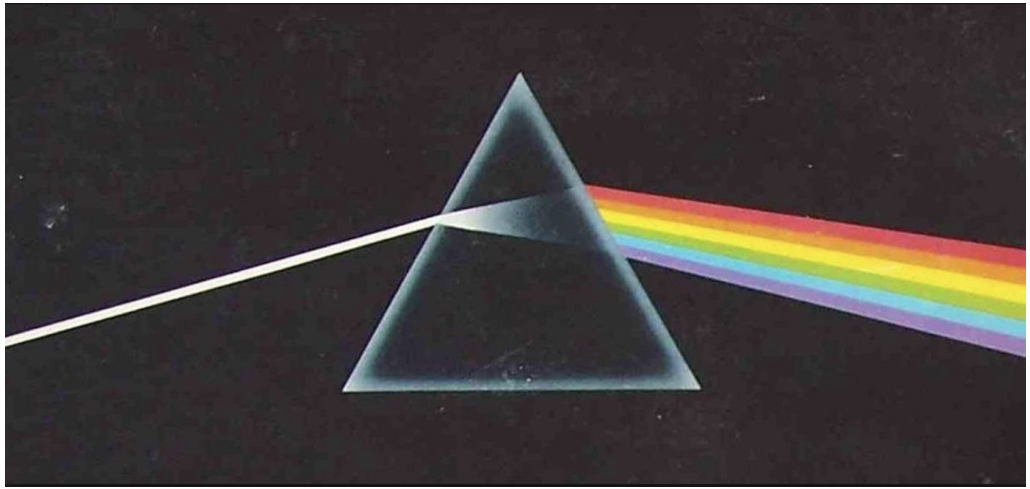
General transformers:

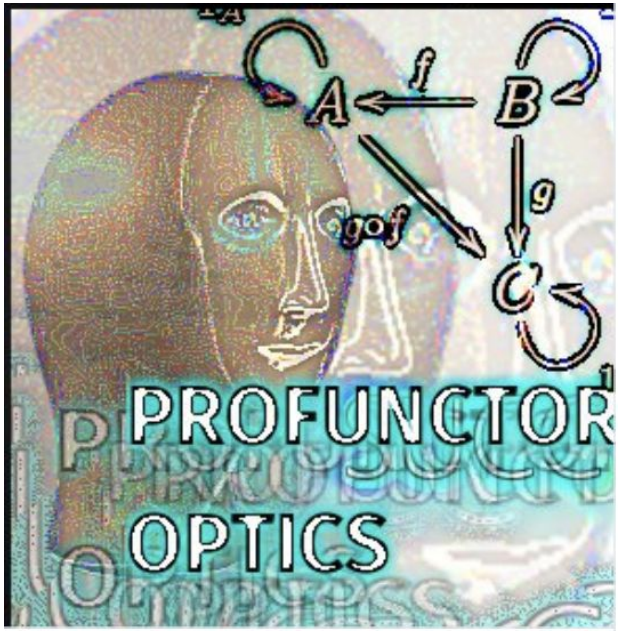
Provide a type-level function from (type A) => (type B)

such that

Given a value-level function (A => E), and a (Query[A]), produce a (Query[B])

Sloppy
A Slick Optics Library





Sloppy
A Slick Optics Library (well, kinda)

Caveats:

Don’t use this in production yet

Not published maven or anywhere else

Presentation Overview

1) The problem
2) Intro to Slick
3) Reusability
4) **Examples (Show me the code!)**
5) The gotchas
6) Summary
7) Fin

Sloppy
A Slick Optics Library (well, kinda)

<Insert demo here>

Sloppy
A Slick Optics Library (well, kinda)

Real world uses:

- RDF-triples-in-RDBMS
- Entitlements / permissions enforcement
- GraphQL virtual schema -> SQL
- Other...

Presentation Overview

1) The problem
2) Intro to Slick
3) Reusability
4) Examples (Show me the code!)
5) **The gotchas**
6) Summary
7) Fin

Slick: The ugly

Shapes everywhere

Slick: The ugly

No CTEs or common subexpression elimination

Monadic join on **Query** likely to lead to runtime heartache

DBIOAction's result type parameter is in the wrong spot :(

IntelliJ highlighting and typechecking FAIL :(

...and more

Slick: The ugly

AST -> SQL compilation is slow

Precompilation is essential for hot-path (but it’s a PITA)

SQL compilation

This is a runtime operation, and is SLOW

Our experience: 50-1000ms

These can be cached once compiled, and with some tricks can be precompiled

SQL pre-compilation

```
object Precompiled {
  /**
    * Create a new `Compiled` value for a raw value that is `Compilable`.
    * This will also precompute (and not just cache) the compiled version of all statements.
    */
  @inline def apply[V, C <: Compiled[V]](raw: V)(implicit compilable: Compilable[V, C], driver: BasicProfile): C =
    using(Compiled(raw)) { c =>
      val d = c.asInstanceOf[CompilersMixin]
      Try(d.compiledQuery)
      Try(d.compiledDelete)
      Try(d.compiledInsert)
      Try(d.compiledUpdate)
    }
}
```

SQL pre-compilation

Precompiled queries tend to be functions from lifted inputs (**Rep[Foo]**) into queries (**Query[R]**)

Presentation Overview

1) The problem
2) Intro to Slick
3) Reusability
4) Examples (Show me the code!)
5) The gotchas
6) **Summary**
7) Fin

Category Theory

Category Theory

If you saw this and thought “this reminds me of **profunctor optics**”, then you get a cookie

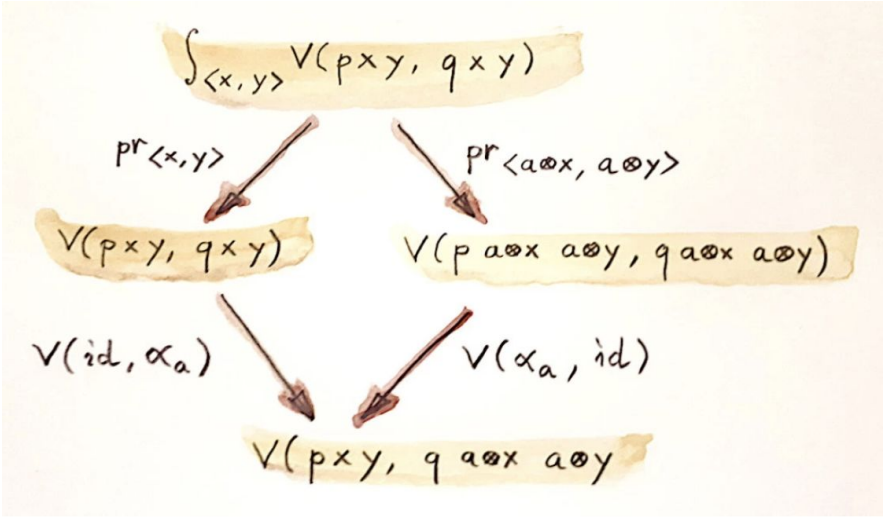


Image apologies to Bartosz

Category Theory

If you saw this and thought “this reminds me of **profunctor optics**” **NO TIME, I’m sorry, but yes**

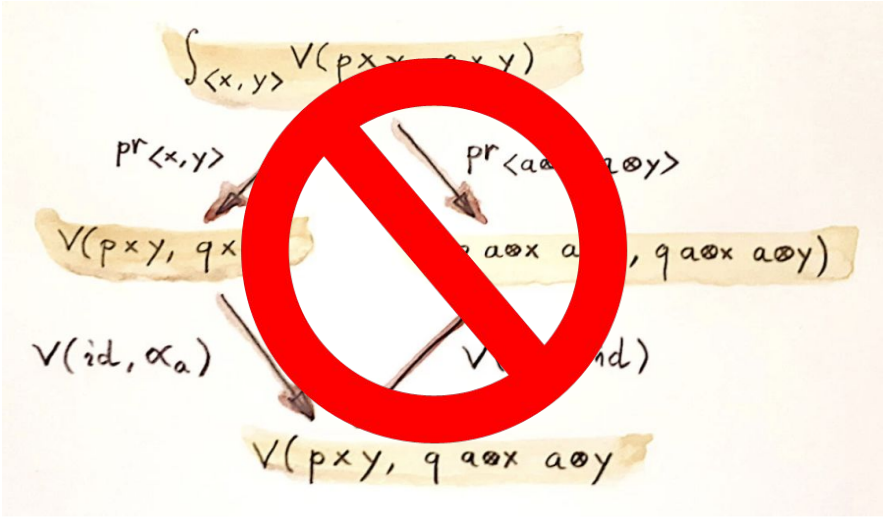


Image apologies to Bartosz

Summary

Slick is pretty cool, check it out

With a little thought and abstraction, your DB interactions can be highly reusable and performant

Check out **Sloppy** and help me build it if you think it's interesting

Special Thanks & Supplemental Materials

Dataset stolen from https://github.com/melaniewalsh/sample-social-network-datasets

Profunctor Optics paper that started it all

Helpful intro to optics with diagrams

Bartosz: profunctor optics talk

Image stolen from the interesting Bartosz on Lenses page

David Spivak's work on Categorical Databases

Thanks! Questions?

https://twitter.com/virus_dave

dave.nicponski@gmail.com

https://github.com/virusdave/sloppy

https://github.com/virusdave/sloppy-example